Appliances energy prediction with Machine Learning:

Comparative Analysis of Algorithms

Adib Md Alim Chowdhury	Caden Quiring	Matthew Muscedere	Tayo Alalade
chowdh52@uwindsor.ca	quiringc@uwindsor.ca	muscede2@uwindsor.ca	alaladej@uwindsor.ca
110036540	110045557	110063159	110038930

Abstract - The UC Irvine Machine Learning Repository Appliances Energy Prediction Data Set consists of 19,735 instances, each with 28 features. Each instance includes an 'Appliances' value, representing the total energy consumption of appliances for that instance. Researchers have utilized this dataset to evaluate the accuracy of various machine learning and deep learning algorithms in predicting appliance energy consumption in a building. The performance of these algorithms is compared under two scenarios: one using default hyperparameter settings and another with optimized hyperparameters. This comparison helps identify the most effective model for accurately predicting energy consumption and assesses the improvements that optimization algorithms can provide.

I. INTRODUCTION AND PRESENTATION OF RESEARCH QUESTION

The Appliances Energy Prediction dataset, available on the UC Irvine Machine Learning Repository, is a popular choice for regression tasks and perfectly aligns with the requirements of our project. This dataset captures indoor conditions using wireless sensor data, averaging measurements over 10-minute intervals to ensure precision. Additionally, it incorporates data from a weather station, merging external weather conditions with the internal environment. This integration allows us to make accurate predictions about appliance energy consumption based on the relationship between indoor and outdoor conditions. For instance, a significant temperature differential between the indoors and outdoors may indicate a high electricity consumption by the house's heater. Our machine learning models aim to recognize and reflect such patterns in the data, ultimately influencing the 'Appliances' value and predictions.

To streamline our dataset, we removed the features 'date,' 'Tdewpoint,' 'rv1,' and 'rv2' as they were considered insignificant for accurate energy predictions. This leaves us with 24 features and the 'appliances' energy column to work with. The dataset exhibits a wide range of energy consumption, spanning from 10 Wh to 1080 Wh, providing ample variability for training our machine learning models.

The Appliances Energy Prediction dataset serves as a valuable resource for research and practical applications, with broad implications across various fields. It has enormous potential particularly in enhancing energy efficiency in buildings. For example, it can be instrumental in identifying energy consumption inefficiencies within a structure. Consider a scenario in a humid environment where a house exhibits low room humidity. By comparing this house to another in the same environment with similar room humidity, one can determine whether the former would benefit from a more efficient dehumidifier. This principle extends to heating, cooling, lighting, and various other aspects of energy use in a building.

Implementing insights drawn from this dataset can yield several benefits, including peak load management, load balancing, reduced carbon footprints, cost savings, optimized HVAC systems, and the integration of renewable energy sources. By enhancing energy efficiency and reducing carbon footprints, this approach can contribute to addressing real-world issues like climate change by alleviating the stress on energy grids

Another use of the dataset can be the research aspect. Many different fields such as energy efficiency, machine learning and data science have researchers who use datasets such as this to train, test and develop new algorithms, methods and models for the prediction of energy consumption. This allows for many different innovations in energy management, consumption and conservation techniques which can then be applied to many residential to industrial settings. More informed decisions on energy efficient technologies and practices can be made from studying datasets such as this to uncover hidden insights and trends. This research can help in shaping future standards/policies, use and development of sustainable energy and environmental conservation. Overall, datasets such as this are great resources for advancing our understanding of how the world consumes energy and help us optimize it in our rapidly changing world.

In machine learning, regression is a supervised learning algorithm used to predict real-valued output based on given features. The primary goal of a regression model is to create a function that maps input features to an output. These features can be either numerical or categorical, while the output represents a numerical value that estimates a dependent variable based on one or more independent variables, which are the features. For instance, in a stock market price prediction model, input data might include various moving averages of stock prices over different time periods, and the output data would be the predicted current stock price. To train a regression model, we start with a labeled dataset containing input data points and their corresponding real-valued outputs. The model learns to map the input data to make predictions by identifying patterns within the data. For example, in the scenario where it's significantly colder outside the house than inside, the machine learning model won't interpret data as humans do but will learn that such conditions typically result in increased energy usage.

Various machine learning algorithms are available for regression predictions, including K-Nearest Neighbors (K-NN), Linear Regression, and Bayesian Ridge. Once a model is trained on the dataset, it can predict the numerical output of new data points. The model takes the features of the new data points as input and uses the mapping function it has learned to predict the output value.

To evaluate the regression model's accuracy, several techniques can be used, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared. A combination of techniques is typically employed to obtain an accurate assessment of the model's performance.

The aim of this research is to explore the feasibility of using machine learning for accurately predicting energy consumption in buildings based on various input features. We tested several machine learning models, including Linear Regression, K-Nearest Neighbors (K-NN), Neural Network (NN), Bayesian Ridge, and Decision Tree Regressor (DTR). To assess their performance, we compared these models with and without hyperparameter tuning using evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-Squared. This evaluation helps us understand their effectiveness in predicting appliance energy consumption in buildings. This study seeks to determine the most accurate models for this task, shedding light on the applicability of machine learning to this problem. The results can provide valuable insights into whether these models can make accurate predictions and whether they can offer practical insights in real-world scenarios.

II. RELEVANT LITERATURE OVERVIEW

Using machine learning for simulation or prediction is a well established process, combined with many datasets like the one we used from UC Irvine Machine Learning Repository, means that we can find many published reports and articles which look to solve the same problem we are looking into. One of these works is “Appliances Energy Prediction Using Random Forest Classifier” by Vinay Vakharia, Sagar Vaishnai and Harshit Thakker. In this report, they used Random Forest, Artificial Neural Network and Support Vector

Machine to train and test in order to compare. They used a dataset that was collected from a low energy consumption house located in Stambruges in Belgium, where the data was divided into 3 subsets (All Records, No lights and Lights). From their experiment, it was observed that Random Forest gave the highest correlation coefficient and the least errors compared to the other 2. It was also mentioned that Random Forest ranked the second fastest of the 3 methods, making it the overall best method.

Machine learning (ML) refers to the application of computer algorithms that can learn patterns and make predictions based on existing data. One practical use of ML is clustering buildings for energy benchmarking, as illustrated in Diagram 2. This process involves various measurements based on actual and predicted results to assess the performance and accuracy of data-driven models. Artificial Neural Networks (ANN) play a significant role in building management systems, enabling automatic control of energy consumption (Kalogirou, 2000; Benedetti et al.). For example, Ben-Nakhi (2004) applied a general Recurrent Neural Network (RNN) to predict the energy consumption profiles of public buildings for upcoming days, utilizing hourly energy consumption data. The goal here is to optimize the management of heating, ventilation, and air conditioning (HVAC) thermal energy storage. However, it's important to note that Support Vector Machines (SVM) have their limitations. One primary drawback is their computation time, which can be quite lengthy, with the complexity growing as the cube of the number of problem samples. This factor can impact their practical application in scenarios where real-time or near-real-time results are required.

III. TOOLS USED

For this research, we chose to implement a selection of machine learning algorithms, including Linear Regression, K-NN, Neural Network, Bayesian Ridge, and Decision Tree Regressor. We conducted these implementations using Python, primarily due to its rich ecosystem of high-level machine learning libraries.

Among these libraries, Scikit-Learn stood out for several reasons. It offers an easy-to-use, consistent API for the selected machine learning algorithms, which simplified our implementation. Additionally, Scikit-Learn boasts excellent documentation, complete with clear examples, facilitating effective utilization of the library. Moreover, Scikit-Learn provides tools for hyperparameter tuning, which we leveraged to optimize the models. Its compatibility with other Python libraries commonly used in machine learning, such as Pandas and Matplotlib, further streamlined our workflow.

Pandas proved invaluable for tasks like loading the dataset from the CSV file, data cleaning, and feature selection. We used it to eliminate unnecessary feature columns and handle missing data. Matplotlib allowed us to visually compare the performance of the models.

To fine-tune our models, we opted for a grid search approach for hyperparameter tuning. This exhaustive method explored a wide array of parameter combinations, aiming to maximize prediction accuracy and assess the models' reliability in real-world scenarios. Grid searches are comprehensive and meticulous, leaving no hyperparameter setting unexamined.

To evaluate model performance, we employed three key metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-Squared (R2). MSE served as an effective measure of regression model accuracy, penalizing larger errors more heavily. This was particularly important as larger errors can have a more significant real-world impact. MAE, less sensitive to outliers, provided a robust performance measure and expressed accuracy in the same units as the target variable (Wh). Lastly, R-Squared gauged the goodness of fit, offering insights into how well the model explains data variance. This standardized measure of model fit allowed us to compare and contrast the performance of different models.

By utilizing these three evaluation metrics in conjunction, we gained a comprehensive understanding of the models' performance and their relative performance when compared to one another.

IV. CHOSEN METHODS TO TACKLE RESEARCH QUESTION

IV.A. LINEAR REGRESSION ALGORITHM(LR)

The Linear Regression algorithm was utilized in researching this dataset due to its simplicity and transparency. The question that was put forward for such a simple algorithm was to see how well it would hold up against the much more complex and computationally intensive algorithms.

The unoptimized algorithm used the provided default settings given by scikit-learn, however grid search was utilized to discover in theory the best hyperparameters for the algorithm. The parameter grid used went as follows:

- fit_intercept: True, False
- copy_X: True, False
- positive: True, False

By utilizing grid search to find the optimized hyperparameters, the results received were as follows:

- fit_intercept: False
- copy_X: True
- positive: False

To summarize, fit_intercept determines whether or not the intercept will be used in the calculations. copy_X determines whether or not X will be copied, and finally positive if true forces all the coefficients to be positive.

IV.B. K-NEAREST NEIGHBOURS ALGORITHM(K-NN)

The K-Nearest Neighbours Algorithm (K-NN) was chosen to see if there was a higher correlation utilizing the neighbours of each query point. It is still one of the less computationally intensive algorithms, but still incredibly effective in the right environment.

The unoptimized algorithm, like the Linear Regression algorithm utilized the default settings provided by scikit-learn. Grid search was once again used, with many more options to choose from this time around:

- n_neighbors: 2, 5, 10, 25
- weights: 'uniform', 'distance'
- algorithm: 'auto', 'ball_tree', 'kd_tree', 'brute'
- leaf_size: 15, 30, 50
- p: 1, 2, 3, 5

Ultimately, after running all possible outcomes from the given parameter grid, grid search produced these as the optimized hyperparameters:

- n_neighbors: 5
- weights: 'distance'
- algorithm: 'auto'
- leaf_size: 15
- p: 1

Each of these hyperparameters have a significant effect on how the algorithm is run.

n_neighbours Is as the name implies; it checks the specified amount of neighbours based on the current query point. It seems as though having too many neighbours skews the data, but having too little leaves too much to the imagination for the algorithm.

weights has two main options; uniform, and distance. Uniform means that all neigbhours are given equal weight. The other option, distance, does the exact opposite, where each neighbour is given a weight based on how close it is to the query point.

Algorithm is used to determine what algorithm will be used to discover the neighbours of a given query point. While all options were provided for grid search to utilize, it's no surprise that it ended up on auto which is an algorithm that decides which of the three neighbour algorithms it should use based on the provided data.

Leaf_size is the amount of leaves that the Ball Tree or KD Tree is allowed to have (if the algorithm decided is one of those two).

P determines the power parameter for the Minkowski Metric. More details on what these optimal changes do will be mentioned in the "Findings" section.

IV.C. NEURAL NETWORK ALGORITHM (NN)

To create an optimized Neural Network a grid search was ran with the following parameter combinations:

- Hidden_layer_sizes: (64, 32), (128, 64), (64, 32, 16), and (64)
- Activation function: relu and tanh
- learning _rate_init: 0.001, 0.01, and 0.1
- Batch_size: 16, 32, and 64
- Early stopping: True and False
- Alpha: 0.0001, 0.001, and 0.01

Hidden_layer_sizes is a fundamental parameter that defines the architecture of a Neural Network. It specifies the number of layers in the network and the number of neurons in each layer. This parameter plays a vital role in shaping the model's ability to learn complex data relationships and in controlling the model's complexity. The number of layers corresponds to the number of integers in the parameter, while each integer represents the number of neurons in a specific layer.

Activation functions are critical elements in Neural Networks as they introduce non-linearity into the model. This non-linearity allows the model to learn complex relationships within the data. Among commonly used activation functions, 'Relu' is a popular choice because it accelerates training and helps the model capture non-linear relationships. 'Tanh' is another option that enables the model to model both positive and negative relationships in the data while ensuring neuron activation during training.

Learning_rate_init is a parameter with significant influence on the optimization process during Neural Network training. It determines the rate at which the weights of the Neural Network are updated throughout the training process. The learning rate is crucial as a too-high rate may cause the optimization process to overshoot optimal weights, while a too-low rate could result in the model getting stuck in local minima and convergence issues.

Batch_size dictates how many data samples the model processes during each iteration in the training process. This choice affects various aspects of training, including memory usage, computation time, stochasticity, and generalization. Smaller batch sizes result in more random weight updates since they are computed on smaller data subsets. This can help the model escape local minima but may lead to noisy updates. Larger batch sizes provide more stable weight updates and a smoother optimization process, but with smaller datasets, it may lead to overfitting.

Early_stopping is a regularization technique used to prevent overfitting. It monitors the model's performance during training and stops the training once performance begins to degrade. This prevents the model from overlearning the training data, thus enhancing its ability to generalize to unseen data. Early stopping not only improves performance but also reduces redundant training, saving time.

Alpha is another method for preventing overfitting. This parameter controls the strength of L2 regularization, which encourages the model to keep the weights of the parameters small. This prevents the model from fitting the data too closely and facilitates better predictions on unseen data. However, using excessively large alpha values can lead to underfitting and poor model performance.

For the final optimized Neural Network, we employed the 'adam' solver/optimizer. While alternative optimization functions could have been explored to find the most optimal one, it's important to note that the grid search, as conducted, already consumed a substantial amount of computational time (two and a half hours). Given this, Adam was chosen as it's one of the most widely used optimizers for training Neural Networks.

Adam offers several advantages, including ensuring that each parameter receives an appropriate update during training. It accelerates convergence, performs bias correction, and proves effective when training Neural Networks on noisy datasets. This choice aligns with the practical considerations of computational efficiency and the proven effectiveness of the Adam optimizer in the context of Neural Network training.

The best best combination of hyperparameters that was found from running the grid search is as follows:

- Hidden_layer_sizes:(128, 64)
- Activation function: relu
- learning _rate_init: 0.001
- Batch_size: 64
- Early stopping: False
- Alpha: 0.001

The choice of hidden layer sizes for the Neural Network revealed that increasing the number of neurons in a two-layer architecture had a positive impact on model performance. However, adding an additional hidden layer didn't provide as much benefit as the increase in neurons. This suggests that the dataset may not contain complex patterns that require the additional layer for effective training. The absence of this complexity may have resulted in overfitting, which would ultimately reduce the model's performance. Consequently, the two-layer architecture with increased neurons is likely the most optimal choice in this context.

Additionally, the grid search determined that the 'relu' activation function outperformed 'tanh' for this dataset. This aligns with the hypothesis that the model was overfitting the data in architectures with more layers. 'Relu' typically allows for greater generalization compared to 'tanh.'

The selection of a learning rate of 0.001, which was the smallest value tested, suggests that the model benefited from a slower convergence rate and smaller weight updates during training. A batch size of 64 was chosen, which enabled more stable weight updates and a smoother optimization process, leading to improved prediction accuracy and potentially faster convergence.

With early stopping set to 'false,' it indicates that the model's performance didn't deteriorate from increased training on the dataset, and it was not overfitting the training data. This aligns with the relatively small alpha value of 0.001, indicating that a moderate level of regularization was applied. These choices collectively reflect a well-balanced approach to building an effective Neural Network for this specific task.

IV.D. DECISION TREE REGRESSOR ALGORITHM (DTR)

Decision Tree models learn using trees where the dataset is split from the root to create leaf nodes until it can't be split any more. It then uses these splits to make its predictions. Decision Tree can be used for both classification or regression where it can give discrete output or continuous output respectively. I have decided to use Decision Tree Regression to compare to the other three learning models for this report. As mentioned before we used the python library scikit-learn and in their website we can check out all the parameters that are available for us to tune and try the models. In total this model has eleven parameters that can be tuned, however I took four of them that I found to be important to use the grid search on while keeping the other ones as default.

In the grid search, I used the following combination of parameters:

- Criterion: friedman_mse, squared_error, absolute_error, poisson
- Max_depth: 2, 5, 10, 15
- Min_samples_split: 2, 5, 10, 15
- Random_state: 2, 5, 10, 15

Criterion parameter is to set the quality measure of the splits. It can be used with different criterias such as "squared_error" which minimizes the L2 loss using the mean of each terminal node, hence useful for decreasing the mean squared error. There is "friedman_mse" which uses mean squared error as the name suggests with Friedman's improvement score for potential splits. "Absolute_error" reduces the L1 loss for the mean absolute error using the median of each terminal node. Lastly, there is "Poisson" which is used to find splits using reduction in poisson deviance.

Max_Depth parameter is used to set the depth of the tree that the model will create in order to learn. The default would be set to None which will allow the nodes to expand until all leaves are pure or till all leaves have less than min_samples_split samples, which is another parameter. This can be set to integers such as 1, 2, 3 etc.

Min_samples_split parameter is used to set how many samples are needed in order to split an initial node of the tree.

Random_state parameter sets the randomness during the training process. The way features are shuffled during splits even when using the best splitting strategy, the parameter determines how the features are shuffled during each of the splits.

IV.E. Bayesian Ridge ALGORITHM (BR)

Bayesian Ridge Regression (Bayesian Ridge) was selected to explore its potential for capturing complex relationships in the data while benefiting from Bayesian principles. It is considered computationally efficient, making it suitable for various applications. For the initial modeling, default settings were used in conjunction with the scikit-learn library. Subsequently, a grid search was conducted to fine-tune hyperparameters, which included the following options:

- max_iter: 100, 300, 500
- alpha_1: 1e-6, 1e-7, 1e-8
- alpha_2: 1e-6, 1e-7, 1e-8
- lambda_1: 1e-6, 1e-7, 1e-8
- lambda_2: 1e-6, 1e-7, 1e-8

After exhaustive evaluation across the parameter grid, the following hyperparameters were deemed optimal:

- max_iter: 100
- alpha_1: 1e-06
- alpha_2: 1e-06
- lambda_1: 1e-08
- lambda_2: 1e-06

Each of these hyperparameters plays a vital role in shaping the Bayesian Ridge model. Max_iter determines the maximum number of iterations for model convergence. Alpha_1, Alpha_2, Lambda_1, and Lambda_2 control the regularization strengths for various parts of the model. These choices are pivotal in crafting a well-performing Bayesian Ridge model for the specific dataset at hand.

V. FINDINGS

Our experiment has provided us with many evaluation scores to analyze and compare. First let us look at the evaluation scores of each model before optimization. Looking at the graphs Mean Squared Error (MSE), Mean Absolute Error (MAE) and R^2 Score, we can see the evaluation scores of the models that were used.

V.A. SCORES BEFORE OPTIMIZATION

Visual representations of the scores are presented in the Appendix section where we can see “Mean Squared Error Graph”, “Mean Absolute Error Graph” and “R^2 Score Graph”.We see that Linear Regression scored 8318 for MSE, 52.5 for MAE and 0.16 for R^2. K-NN scored 5963 for MSE, 36.7 for MAE and 0.4 for R^2. NN scored 8431 for MSE, 57 for MAE and 0.15 for R^2. Bayesian Ridge scored 8317 for MSE, 52.5 for MAE and -0.15 for R^2. DTR scored 9423 for MSE, 40.6 for MAE and 0.17 for R^2. Looking at these scores we can see clear performance differences between some of the methods while others are similar.

For MSE, K-NN is doing the best out of all the methods by a significant margin. LR, NN and BR have similar scores and are almost tied for second lowest. DTR comes last with the highest MSE score.

For MAE, again K-NN is doing the best by a significant margin. DTR comes second lowest score while LR, NN and BR are tied for last place as their scores are the highest by a significant margin.

For R^2 score once again following the trend, K-NN is the best as it results in the highest score. NN, LR and BR come second highest in terms of scoring. Following that DTR scores the lowest positive value as it comes last in the ranking.

V.B. SCORES AFTER OPTIMIZATION

Since we used grid search to tune in our model, they were optimized and some models resulted in better scores in some places.

“Mean Squared Error Graph”, “Mean Absolute Error Graph” and “R^2 Score Graph” in the Appendix section can be used to see how the optimized models compare to non-optimized models, we see that LR and BR didn’t have a significant improvement in terms of scoring. K-NN was the best out of the models we compared before the optimization and it got even better through the optimization. For MSE and MAE we noticed an decrease of 27.97% and 18.20% respectively, making it go even closer to 0. For R^2 score we see an increase of 41.25% pushing it even closer to 1. NN saw some improvements where MSE decreased by 10.36% and even more significantly for MAE where

it decreased by 19.97%. For R^2 score it increased by 55.44%. For DTR, we saw improvements for MSE and R^2 where it decreased by 18.21% and increased by 293.81% respectively. However, we saw an increase for MAE by 7.01% which was not expected as optimization is supposed to decrease the absolute error, so it may need more optimization work for example with the max depth being set to the default setting 'None'.

V.C. STANDARDIZING THE DATA FOR THE NEURAL NETWORK

Standardizing the data for the Neural Network yielded a notable improvement in model accuracy. In the case of the non-optimized Neural Network, the Mean Squared Error (MSE) decreased from 8431 to 7453, the Mean Absolute Error (MAE) decreased from 57 to 48, and the R-Squared (R2) increased from 0.15 to 0.25. For the optimized Neural Network, the MSE decreased from 7557 to 6132, the MAE decreased from 45 to 44, and the R2 increased from 0.23 to 0.38. These improvements marked a significant enhancement in the Neural Network's performance, positioning it as the second-best model in terms of MSE and R2 score. In the case of MAE, it was slightly outperformed by the Decision Tree Regressor (DTR).

These results align with expectations, as Neural Networks typically demonstrate improved performance when the data is standardized. The reduction in error metrics and the increase in the R2 score indicate that standardization contributed to a more accurate and reliable prediction model.

VI. CONCLUDING REMARKS

After the experiment was completed, we noticed some of the models stand out more than the others as seen in the previous section. One of them being the Optimized K-Nearest Neighbour model which is doing the best for all 3 of the performance evaluation methods. It scores 4295 on Mean Squared Error which is the lowest, 30 on Mean Absolute Error which is the lowest and 0.57 on R^2 Score which is the highest. Even though these results are the best out of the models we used, they still indicate an overall poor performing model. We want to go as close as we can to 0 for MAE and MSE, and as close as we can to 1 for the R^2 score. This means that there is still a lot of work in terms of refinement needed to improve the performance of these models and also maybe find better models. In conclusion, many regression learning models can be used for the prediction of application energy however there is a significant amount of results that show us that testing out and comparing many different models gives an idea of how good they are performing comparatively to choose the best one, and also that optimizing by tuning the paraments of the models using grid search can result in a significant increase in the results that they produce.

VII. FUTURE WORK

Further research in predicting appliances' energy use in a specific building can take several directions. Firstly, exploring additional machine learning models beyond those initially tested, such as Linear Regression, K-Nearest Neighbors, Neural Network, Bayesian Ridge, and Decision Tree Regressor, may reveal more accurate models tailored to this specific problem. In addition, conducting more extensive hyperparameter tuning or using optimization techniques, like Bayesian optimization, can fine-tune the models to enhance their accuracy.

However, it's crucial to address the computational limitations, and obtaining increased computational resources or utilizing distributed computing could enable more comprehensive grid searches and reduce time constraints. Another avenue for improvement involves gathering a larger dataset to provide the models with more parameters for training, thereby allowing them to learn more intricate data patterns and better generalize information, ultimately reducing susceptibility to outliers.

In the evaluation process, considering a wider array of metrics, such as Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), or custom loss functions, can offer a more thorough assessment of the models' performance. Lastly, it's essential to explore potential real-world applications for these models beyond energy use prediction, thereby maximizing their utility in fields such as energy efficiency, load management, cost savings, and environmental conservation.

VIII. REFERENCES

[1] V. Vakharia, S. Vaishnani, and H. Thakker, "Appliances energy prediction using random forest classifier," SpringerLink, https://link.springer.com/chapter/10.1007/978-981-15-8704-7_50 (accessed Oct. 19, 2023). Candanedo,Luis. (2017). Appliances energy prediction. UCI Machine Learning Repository. https://doi.org/10.24432/C5VC8G.

[2] S. Seyedzadeh, F. P. Rahimian, I. Glesk, and M. Roper, "Machine learning for estimation of building energy consumption and performance: A review - visualization in engineering," SpringerLink, https://link.springer.com/article/10.1186/s40327-018-0064-7 (accessed Oct. 19, 2023).

[3] "Sklearn.tree.decisiontreeregressor," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html (accessed Oct. 19, 2023).

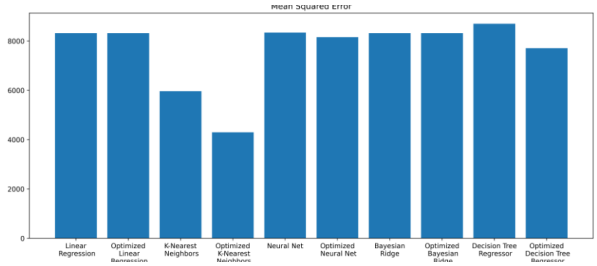
[4] "Sklearn.neighbors.kneighborsregressor," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html (accessed Oct. 19, 2023).

[5] "Sklearn.neural_network.Mlpregressor," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html (accessed Oct. 19, 2023).

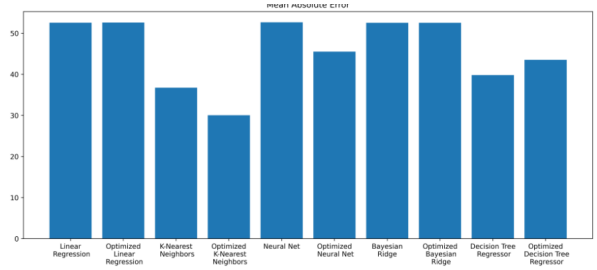
[6] "Sklearn.linear_model.Bayesianridge," scikit, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html (accessed Oct. 19, 2023).

IX. APPENDIX

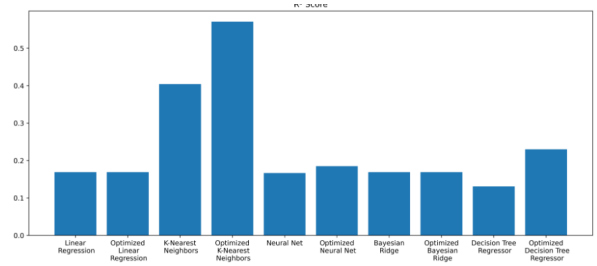
Mean Squared Error Graph



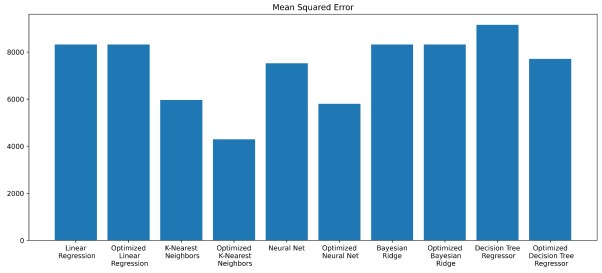
Mean Absolute Error Graph



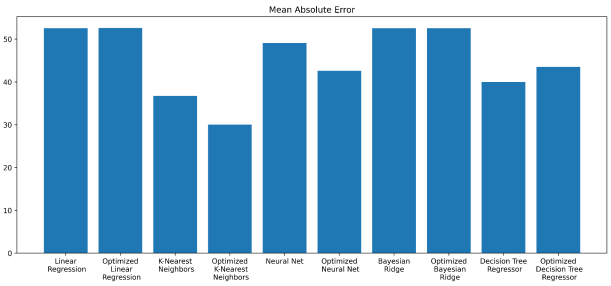
R^2 Score Graph



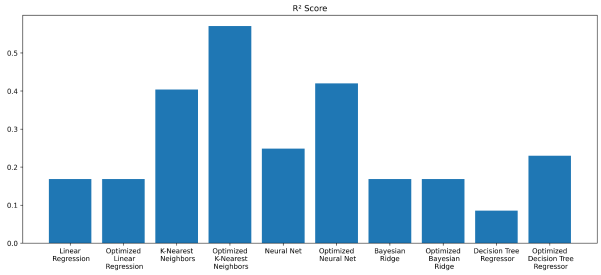
Mean Squared Error Graph With Standardization Applied for the Neural Network



Mean Absolute Error Graph With Standardization Applied for the Neural Network



R^2 Score Graph With Standardization Applied for the Neural Network



X. GROUP CONTRIBUTIONS

Matthew Muscedere:

- Found dataset
- Developed Neural Network grid search and other Neural Network code
- Wrote abstract section
- Wrote paragraphs 1-2, worked on 3, and wrote 5-7 in introduction section
- Wrote Tools used section
- Wrote IV.C
- Wrote Future work section
- Wrote V.C

Adib Md Alim Chowdhury:

- Worked on intro paragraph 3
- Wrote intro paragraph 4
- Wrote literature overview paragraph 1
- Wrote IV.D. DTR
- Wrote V.A. and V.B
- Wrote VI
- Wrote code for DTR
- Wrote code for grid search and parameters
- Wrote code for Optimized DTR
- Organized Appendix

Caden Quiring:

- Prepared the dataset.
- Set up the GitHub repository.
- Wrote grid search code for Linear Regression and K-NN.
- Wrote the entirety of reg_algos.ipynb with the exception of normalizing data and any algorithm in the models dictionary that wasn't related to linear regression or K-NN.
- Wrote README.md.
- Wrote IV.A.
- Wrote IV.B.

Tayo Alalade:

- Wrote Relevant literature paragraph
- Wrote bayesian ridge paragraph
- Wrote references paragraph
- Wrote code for bayesian ridge
- Worked on formatting of doc
- Wrote code for grid search and parameters
- Wrote code for Optimized Bayesian Ridge